

1.React + Spring Boot todoList

React = components + features

Components:

1.jsx 是 js 的扩展在 js 文件里有类似 html 的 code, 负责 view

2.js 负责逻辑 3.css 负责 style 4.state 存数据 5.props 传递数据

virtual DOM: React's representation of DOM in memory, 在 memory 所以 make change 很简单。React 先找到 Virtual DOM 和 DOM 有什么区别, 再把不同的地方在 DOM 上更新了

Features: 1.Routing 2.Forms & Validations 3.Rest API Calls 4.Authentications

介绍:

这是一个用户可以注册登录去做 todolist 的网站, 里面可以新增 修改 删除 todo。每个 todo 都有 id 任务描述 目标日期 是否完成这些内容

细节:

用 BrowserRouter 去到不同的路由(有登录, 登出, 欢迎页, todoList 页), 用 switch 确保只去到一个路由, 如果路由地址不对就会去到一个 ErrorComponent。每页都会有 header 和 footer, header 能被点击去到其他路由, 但是会用 sessionStorage(在 inspect 里 Application 里的 storage 下, 里面是 key value 形式, 把用户用户名作为 value 存。不用 localStorage 是因为他没有 expiration time)去(setItem(登录) removeItem(登出) getItem(判断是否是登录的状态))判断用户的状态, 只有在登录状态下才会有 home, todolist 和登出的元素在 header, 只有登出状态下才有 login 这个 Link。

1.Get 用户的 todolist: 首先数据在后端获取然后映射到一个 URL 上, 把用户名设为 path variable, 之后在前端的 componentDidMount 用 axios 获取 response, 也就是用户的 todos 来设置指定用户的 state。这个放大用于每次 render 完 delete 完

2.Delete 用户 todo: 点击 button 的时候传 todo 的 id, 之后删除的方法里会用用户名和 id 用 axios 带上指定 URL(用户名和 id 作为 path variable)去 delete。后端里 mapping 这个 url 的方法(API)就会执行去删除数据。同时前端页面上会重新 get 用户的 todolist 来展现删除后的 todoList

3.update/add 用户 todo: 点击更新的 button 后会传 todo 的 id 然后带上这个 id 去到一个新的路由去改(this.props.history.push(`/todos/\${id}`))这个 todo, 在新的页面里可以改 todo 的描述和目标日期, 这里用了 Formik 这个包去对用户输入数据作一个验证, 之后在点击保存这些修改的按钮的时候会把输入内容封装成一个 object 作为 todo, 之后看当前 state 的 id 是不是 -1, 如果是的话说明没有这个 todo, 是新增, 就用 axios 和 URL 去调用后端新增的 API, 否则就调用更改的 API。后端里也是判断 todo 的 id, 如果是 -1 就新增, 如果不是就把原来的删了再新增。更改的时候 URL 要带上 todo 的 id, 新增的时候不用。更改完后会用 history.push 回到 todoList 界面

Security:

在后端加上 jpa 依赖并且 import jpa 的方法实现(网上找的 framework),

在前端登录的时候会用 axios 的带上账号密码(header)的 post 请求(也就是 API_URL/authenticate(spring 里配置的),

再带上 header),(用 BCryptPasswordEncoder 的 encode 方法对用户的密码加密, 底

层大概是加点 salt 再转换成 64 进制)来得到(靠 jpa 方法) **JWT** (JSON Web Token), 之后的 request 都会用这个 JWT (Jott), 得到 token 后先在 sessionStorage 里放 ('authenticatedUser', username), 之后会去用 axios 的 interceptor(可以在 request 和 response 被 then 和 error handle 前处理 request 和 response, 这里就是用来带上 header) 来把 'Bearer' + token 放到配置的 header 里。然后登录到欢迎界面

```
setupAxiosInterceptors(token) {  
  axios.interceptors.request.use(  
    (config) => {  
      if (this.isUserLoggedIn()) {  
        config.headers.authorization = token  
      }  
      return config  
    }  
  )  
}
```

token 也有过期时间,过期时间快到了可以发 get 请求(API_URL/refresh)来更新 token。对于授权其他请求(put delete get post), 只要在 header 里加上 Bearer token 就行。每个请求都会去到 JWT authentication token filter

在后端 pom 里加上 spring-starter-security 依赖, (这样去到任何页面前都会有个 form 要输入账号密码), 这个账号密码可以在 resources 里配置成自己的, 再加上 spring security configuration(extends WebSecurityConfigurationAdapter)来 disable CSRF 和允许没有被 authenticate 的 HTTP 里带有 Options 的 URL(因为在完整的请求被发送之前会有个请求 call options 请求去看有没有许可 permission)最后在前端每次 login 的时候都会去做认证, 也就是用 axios 带上账号密码组合加密(base64)后的 token 去请求到后端指定的 URL, 之后再登录需要做的事情 -- 跳转到欢迎界面。Basic auth 就是 axios 带上 header 这个参数, 里面是 Basic + encoded 账号和密码(encode 用 window.btoa() -- base 64 encoding), 这里是用 axios 的 interceptors 来实现的

Basic auth 的缺点: 1.token 没有过期时间, 别人可以一直用 2, token 里没有用户细节

防止用户输入网址就跳过登录就到 todolist 界面: 创建一个新的 component, 里面判断也是用 sessionStorage 的 component 来判断用户是否登入。如果登入就用...this.props 传入所有参数(path, component), 来去到指定路由。之后对于那些要防止用户直接输入网址的路由, 我们可以用这个 component

JWT(JSON Web Token) 确保服务的安全性, 使其他用户不能输入 URL 就访问到别人的 todolist。好处: 1.可以设置过期时间 2.可以包含用户细节(header, payload-用户名字, 创建时间等等, verify signature), 包括可以设置不同的加密算法

知识点:

1.webservice application request 到 web service, web service response。之间靠的是 XML, JSON 这样的 data format, 具体用什么要看 service definition
service requestor 把 message 放到 MQ, service Provider 会接受并产生 response 放回到 MQ, requestor 接受 response

2.REST: 1. Data Exchange Format 没限制 JSON 比较 popular 2.transport 只用 HTTP 3.service definition 用 swagger 这种

3.dispatch servlet 接受 bean 的 request, 也就是看 URL 和 method, dispatch servlet 会找合适的 controller 去执行。之后得到 bean 再去 invoke conversion, 把他转变为 JSON 再把 response 发回

4. Servlet 和 JSP

1、不同之处在哪？

Servlet 在 Java 代码中通过 `HttpServletResponse` 对象动态输出 HTML 内容

JSP 在静态 HTML 内容中嵌入 Java 代码，Java 代码被动态执行后生成 HTML 内容

2、各自的特点

Servlet 能够很好地组织业务逻辑代码，但是在 Java 源文件中通过字符串拼接的方式生成动态 HTML 内容会导致代码维护困难、可读性差

JSP 虽然规避了 Servlet 在生成 HTML 内容方面的劣势，但是在 HTML 中混入大量、复杂的业务逻辑同样也是不可取的

MVC 模式（Model-View-Controller）是一种软件架构模式，结合 servlet 和 JSP，扬长避短，把软件系统分为三个基本部分：模型（Model）、视图（View）和控制器（Controller）

Controller——负责转发请求，对请求进行处理

View——负责界面显示

Model——业务功能编写（例如算法实现）、数据库设计以及数据存取操作实现在 JSP/Servlet 开发的软件系统中，这三个部分的描述如下所示：

过程：

Web 浏览器发送 HTTP 请求到服务端，被 Controller(Servlet) 获取并进行处理（例如参数解析、请求转发）

Controller(Servlet) 调用核心业务逻辑——Model 部分，获得结果

Controller(Servlet) 将逻辑处理结果交给 View（JSP），动态输出 HTML 内容
动态生成的 HTML 内容返回到浏览器显示

MVC 模式在 Web 开发中的好处是非常明显，它规避了 JSP 与 Servlet 各自的短板，Servlet 只负责业务逻辑而不会通过 `out.append()` 动态生成 HTML 代码；JSP 中也不会充斥着大量的业务代码。这大大提高了代码的可读性和可维护性。

5. axios 可以异步处理请求 `asynchronous`，用 `axios` 可以处理 controller 的 URL，只要传 URL 进去就可以（用户名作为 `path variable`），之后可以对 `response.data` 等进行处理，比如设置为 `state` 的值，如果 controller 里 `throw error` 也一样处理 `--error.data...`

6. 执行顺序：`constructor -> render -> componentDidMount -> render`（如果改变 `state` 就要再 `render`）`-> ComponentWillUnmount`（view 里 component 被 `remove` 了，比如切换页面了）`componentShouldUpdate` 如果 `return false` 就不会更新 view（`render` 不会被 `call` 第二次），但 `state` 改变了

7. JPA 的替代品：问题：如何定义 `object` 和 `table`

JDBC query: `update todo set user=?, desc=?, target_date=?` 如果是 `select` 就会返回一个结果，要对结果自己 `mapping` 到 `bean`

myBatis 可以用 XML 做这样相同的事情，而且 `mapping` 更简单 `user=${user} {}` 里是 `bean`

8. JPA (Java persistence API)（一个 `interface`，`Hibernate` implements JPA，是一个具体实现）可以允许我们定义 `class` 和 `table` 的 `mapping`，可以用 `SQL/JPQL` 去 `query` 数据库。有个 `entityManager`（允许保存 `data` 到数据库）

注释有 `@Entity`（要存到 `table` 就要加）`@ManyToMany`

@Id 说明是 primary key

dao service(有 insert get 等)里 @Transactional Spring 创建和 class implements 一样 interface 的代理(要去对数据库做修改) @Repository 说明要和数据库 interact

persistence context: a synchronizer object that tracks the state of a limited set of Java objects and makes sure that changes on those objects are eventually persisted back into the database. 而 entityManager 不行, 只负责创建时候

JPA 需要 default constructor

SLF4J(simple logging facade for java)

9.H2(in-memory 数据库), 不同于 Mysql, Oracle, H2 不用安装, 直接加了依赖了就能被创建, 程序 run -> 表、schema 自动创建 -> 数据自动被 insert -> 去 localhost:8080/h2-console 可以看见 data

application stop -> 整个数据库销毁, 从内存里 remove

这些因为 auto configuration: spring boot 知道是否连接了 in-memory 数据库, 会自动初始化 schema 和删除 schema(application stop)。hibernate JPA auto configuration 自动配置了 entity 管理 data source 和一个 transaction manager

JPA repository interface 有很多方法可以更好地管理 entity, 有 save(create 新的) findby findAll

具体操作: 运行后在日志里找 spring.datasource.url 之后在 localhost:8080/h2-console 网站里输入 url 就可以对数据库进行操作

2.React 汉堡点餐网站

介绍:

这是一个用于汉堡店的点餐网站, 大体上说用户可以通过邮箱和密码注册登录后按照自己喜好来定制汉堡, 比如通过点击来加减各种配料的数量, 然后相应的汉堡模型也会及时地显示在页面上。点完了之后用户需要输出他们的信息, 联系方式, 地址和配送方式等。最后这个订单会被存在每个用户各自的数据库里, 用户也可以通过导航栏里点击订单来查看订单并且还有登出登入的按钮

细节:

1. 在 index.js 里用 redux 组合 burgerBuilder, order, auth 三个 reducer, 创建 store 去放这个总 reducer, 并且 redux 的 thunk 去实现 middleware。用连接 React 和 redux 的 Provider 去连接然后在里面放 react-router-dom 的 BrowserRouter, 最后在里面放进 App。

2. 在 App 里放进 checkout, orders, auth, logout, /这几个 Route, 用 react-router-dom 里的 Switch 去实现只运行第一个匹配的 Route, 这样用户点击的时候就能到正确的 route

3. 在每个路由里, 分别用了不同的 redux 的 reducer 去实现。通过引入一个总 file(index.js), 里面有各种方法, 每个方法会去派遣对应的 reducer 里的函数去改变一些变量或者实现一些响应。比如加减 ingredients, 认证用户等等。同时在一些函数使用 firebase 来把存放数据和用户标识。

防止刷新就要重新验证: 在 App 这个总 file 里的 componentDidMount 函数里做检查, 如果检查到本地储存了用户 token 就再从本地存储获取用户 id 来登录否则到登出的路

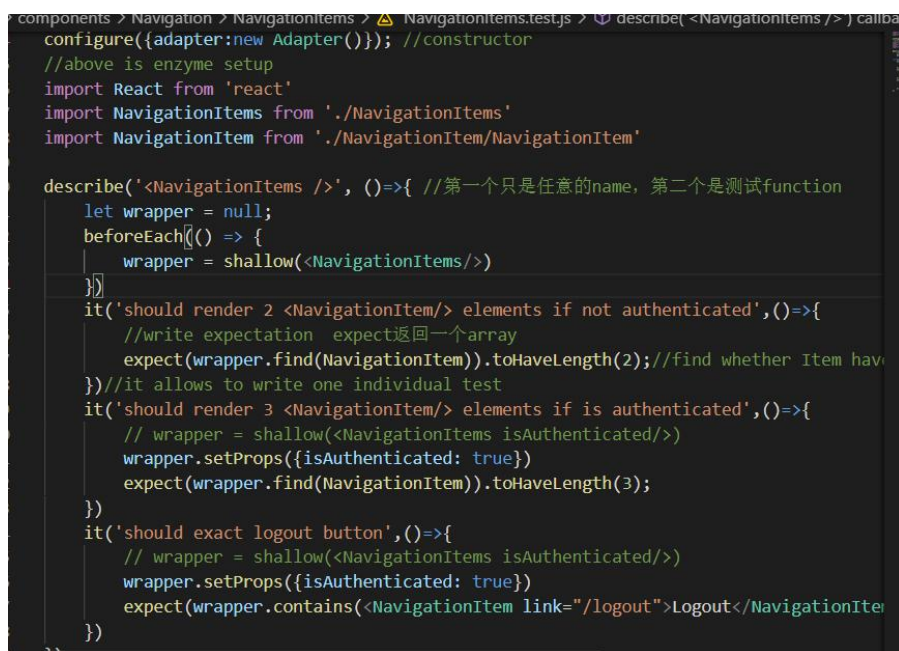
由，这样刷新就自动登录了

登录：首先用户点击 switch to signIn，状态里 signIn 变量变成 true，之后用户输入完邮箱密码点 submit 按钮就会设置状态里的邮箱和密码变量，然后到了 actions 文件夹的文件里检查这个 isSign 变量，axios 根据是不是登录还是注册 post 不同的 url 来提交不同数据，服务端会根据客户端传过来的用户 id 查询数据库，获取 secret，然后重新生成一份 token，之后比较客户端传过来的 token 和服务端生成的 token 比较，相等的话返回数据到客户端，然后设置定制汉堡的路由到定制汉堡页面，不相等的话界面上就会显示密码错误信息

困难：在加完料的时候用户点击 continue 会弹出一个订单总结，上面每个料的数量包括价格，这时候用户有两个选项，一个是点取消就是返回继续加料，一个是继续去到结账页面。我遇到的难点是怎么简单地处理用户点击取消的函数里把页面回到加料页面，点击继续的函数里把页面设置成要跳转的页面。

解决：先打印出当前的 props，然后在检查页面里看 BOM(浏览器对象模型)里的 history 对象，history 里面又有很多的属性可以用，这里我就用了 this.props.history.goBack() 去回到之前加料页面，然后用了 this.props.history.replace(里面加上结账页面的相对路径)去到结账页面

测试：单元测试



```
components > Navigation > NavigationItems > NavigationItems.test.js > describe(<NavigationItems />) callback
configure({adapter:new Adapter()}); //constructor
//above is enzyme setup
import React from 'react'
import NavigationItems from './NavigationItems'
import NavigationItem from './NavigationItem/NavigationItem'

describe('<NavigationItems />', ()=>{ //第一个只是任意的name, 第二个是测试function
  let wrapper = null;
  beforeEach(() => {
    wrapper = shallow(<NavigationItems />)
  })
  it('should render 2 <NavigationItem /> elements if not authenticated',()=>{
    //write expectation expect返回一个array
    expect(wrapper.find(NavigationItem)).toHaveLength(2); //find whether Item have
  }) //it allows to write one individual test
  it('should render 3 <NavigationItem /> elements if is authenticated',()=>{
    // wrapper = shallow(<NavigationItems isAuthenticated />)
    wrapper.setProps({isAuthenticated: true})
    expect(wrapper.find(NavigationItem)).toHaveLength(3);
  })
  it('should exact logout button',()=>{
    // wrapper = shallow(<NavigationItems isAuthenticated />)
    wrapper.setProps({isAuthenticated: true})
    expect(wrapper.contains(<NavigationItem link="/logout">Logout</NavigationItem />)).toBe(true);
  })
})
```

知识点：

1. axios: 基于 Promise 用于浏览器和 NodeJs 的 HTTP 客户端，符合 ES 规范
2. Promise 是异步编程的一种解决方案
3. 把组件看成一个函数，那么他接受 props 作为参数，内部由 state 作为函数的内部参数
4. JSX 是 JS 的语法扩展，ES6 是 JS 下个版本标准
5. 相关知识：为什么要引入(import)React，但在代码里没有用，import React from 'react' 实际上是

React.createElement(component, props, ...children)，JSX(Javascript XML)为此提供了语法糖(指计算机语言中添加的某种语法,这种语法对语言的功能并没有影响,但是更方便程序员使用，

相当于成语，用更简练的语言表达较复杂的的含义，提升交流效率),所以只要 `import React` 就行

6. DOM 就是文档对象模型。在这个文档对象里，所有的元素呈现出一种层次结构，就是说除了顶级元素 `html` 外，其他所有元素都被包含在另外的元素中。VDOM 虚拟 DOM 是真实 DOM 的内存表示。与真实 DOM 同步比如通过 `ReactDOM`

7. 类组件和函数组件的区别：类组件不能改自己的 `props`，但能改 `state`，`redux` 传进来 `state` 到 `props` 里，父组件改变，子组件视图会更新。函数组件接收一个 `props` 返回一个 `React` 元素。函数组件性能高，因为类组件使用时要实例化。类组件有 `this` 和生命周期

8. `render` 函数不往 DOM 树上渲染或者装载内容，只是返回 `JSX` 对象，由 `React` 库来根据这个返回对象决定如何渲染

9. `redux` 是一个应用数据流框架，主要是解决了组件间状态共享的问题，原理是集中式管理，主要有三个核心方法，`action`，`store`，`reducer`，工作流程是 `view` 调用 `store` 的 `dispatch` 接收 `action` 传入 `store`，`reducer` 进行 `state` 操作，`view` 通过 `store` 提供的 `getState` 获取最新的数据，`redux` 中只能定义一个可更新状态的 `store`，`redux` 把 `store` 和 `Dispatcher` 合并,结构更加简单清晰

10. 新增 `state`，对状态的管理更加明确，通过 `redux`，流程更加规范了，减少手动编码量，提高了编码效率，同时缺点时当数据更新时有时候组件不需要，但是也要重新绘制，有些影响效率。一般情况下，我们在构建多交互，多数据流的复杂项目应用时才会使用它们

11. `redux` 缺点：一个组件所需要的数据，必须由父组件传过来当一个组件相关数据更新时，即使父组件不需要用到这个组件，父组件还是会重新 `render`，可能会有效率影响，或者需要写复杂的 `shouldComponentUpdate` 进行判断。

- 将回调函数的参数作为与回调函数同等级的参数进行传递



12.

13. 生命周期函数和组件地生命周期：

组件第一次挂载，获得父组件的 `props` 和初始的 `state`，接着经历 `ComponentWillMount`, `render`, `ComponentDidMount`，挂载完成后组件的 `props` 和 `state` 的任意改变都会导致组件进入更新阶段，`props` 改变进入 `ComponentWillReceiveProps`，再进入 `ComponentShouldUpdate` 判断是否要更新，如果是 `state` 改变则直接进入 `ComponentShouldUpdate` 判断，默认为 `true`，要更新的话进行 `WillMount`, `render`, `DidMount` 三个阶段，不更新组件继续进行，挂载时，进入 `ComponentWillUnMount`，之后进行卸载

14. *初始渲染阶段：*这是组件即将开始其生命之旅并进入 `DOM` 的阶段。

15. *更新阶段：*一旦组件被添加到 `DOM`，它只有在 `prop` 或状态发生变化时

才可能更新和重新渲染。这些只发生在这个阶段。

16. ***卸载阶段:**这是组件生命周期的最后阶段,组件被销毁并从 DOM 中删除。

17. **componentWillMount:** 在渲染之前执行,用于根组件中的应用程序级别配置。

18. **componentDidMount:** 仅在客户端的第一次渲染之后执行。这是 AJAX 请求和 DOM 或状态更新应该发生的地方。此方法也用于与其他 JavaScript 框架以及任何延迟执行的函数(如 `setTimeout` 或 `setInterval`)进行集成,在这里使用它来更新状态,以便我们可以触发其他生命周期方法。

19. **componentWillReceiveProps:** 只要在另一个渲染被调用之前更新 props 就被调用。当我们更新状态时,从 `setNewNumber` 触发它。

20. **shouldComponentUpdate:** 确定是否将更新组件。默认情况下,它返回 `true`。如果您确定组件在状态或道具更新后不需要渲染,则可以返回 `false` 值。这是提高性能的好地方,因为如果组件收到新的道具,它可以防止重新渲染。

21. **componentWillUpdate:** 在由 `shouldComponentUpdate` 确认返回正值的优点和状态更改时,在重新渲染组件之前执行。

22. **componentDidUpdate:** 通常用于更新 DOM 以响应属性或状态更改。

23. **componentWillUnmount:** 它将用于取消任何传出的网络请求,或删除与该组件关联的所有事件侦听器。

React Context:当你不想在组件树中通过逐层传递 props 或者 state 的方式来传递数据时,可以使用 Context 来实现跨层级的组件数据传递。

24. **props 介绍 Props** 是 React 中属性的简写。它们是只读组件,必须保持纯,即不可变。它们总是在整个应用中从父组件传递到子组件。子组件永远不能将 prop 送回父组件。这有助于维护单向数据流,通常用于呈现动态生成的数据。

25. 箭头函数

```
//General way
render() {
  return(
    <MyInput onChange = {this.handleChange.bind(this)} />
  );
}
//With Arrow Function
render() {
  return(
    <MyInput onChange = { (e)=>this.handleChange(e)} />
  );
}
```

26. }

2. GroundRating

1. Express 是 NodeJS Web 应用框架,提供了很多特性和 HTTP 工具,比如路由处理,提供中间件(可以修改请求和响应对象),来帮助创建 Web 应用

2. 在 app.js 文件里先连接 mongoose 然后加载(require)express,之后用 use 去到不同的路由,然后在不同的路由里分别执行不同的操作,比如 get 到了 new 的 route 就去执行创建帖子的页面的模块。在每个页面几乎都用了 ejs-mate 去统一页面的布局,比如导航栏,注脚和一些 flash

知识点:

1. 哈希密码是在用户密码后面加 salt(加强隐蔽性), 也就是随机的值, 再哈希, 这也别人即使通过哈希完的值去暴力配对密码也不能知道正确的密码, 破解难度大大增加

2. Bcrypt 是这个哈希 function, 同一个密码每次生成的 hash 都是不一样的
在加密的时候, 先随机获取 salt, 然后跟 password 进行 hash。在下一次校验的时候, 从 hash 中取出 salt, salt 跟 password 进行 hash, 得到的结果跟保存在数据库的 hash 进行比较。

BCrypt 比 MD5 更安全, 但加密更慢

1. EJS 是一个 JavaScript 模板库, 用来从 JSON 数据中生成 HTML 字符串。

2. npm 是 Java 的包管理程序, 提供 JS 的各种第三方框架的下载

3. Node.js 是一个后端的 js 运行环境, 用来解释 js 代码

4. ajax 的全称是 Asynchronous Javascript+XML 异步传输+js+xml, 点一下不会刷新页面

5. 出现 404 (链接指向的网页不存在) 的排查

一般使请求的 URL 错了

先检查 url 有没有拼对

然后再一些链接前后加一些 console log 看看去找哪里除了问题

```
app.use((req, res) => {  
  res.status(404).send('404 not found!!!')  
})
```

6. JQuery 与 react 不同, 是事件驱动, 其他是数据驱动, 他的 UI 里还有交互逻辑, 所以逻辑混乱, JQuery 操作真正的 DOM, React 根据 virtual DOM 来修改真正的 DOM, JQuery 简化 DOM 操作以及动画, 降低了 js 门槛

7. password 是在 app.js 里使用, 提供了很多函数简化注册流程, 会 hash 密码

8. 用响应的 params 得到每个帖子的 id, 用 findByIdAndDelete 删掉帖子

9. mongoose 创建 schema(定义变量以及他的类型), 创建 model 然后连接到数据库

10. 把图片传到 cloudinary, 然后把图片的文件名和 url 传到对应的帖子, 再在网站里展示出来

登录: 首先设置响应的 user 为请求的 user, 根据 user 存不存在来在导航栏上显示登录或注册和登出按钮, 用户点击不同的按钮会到不同路由, 不同路由有不同界面, 在登陆界面用户输完验证信息点登录的话把用户信息通过 passport-Mongoose/password 的一些方法传到数据库检查对不对然后会对应再到不同界面

困难:

解决:

测试: Postman

XHR: 一种浏览器 API, 简化了异步获取信息的过程

从浏览器导入 request 到 postman: 检查页面里选网络里的 XHR - 勾选 preserve log - 完成操作 - 右键 Name 里发送了的请求 - 选 copy-copy as cURL - 到 postman 点左上角的 import - 选 paste raw text - 复制进去 - 要填的信息都有了

cookie: 上一个请求返回的 cookie 会加到下一次的请求中

cookie 是以 header 的值发送的, postman 提供了查看这些值的功能, 值里打包了很多东西
troubleshooting: 1.检查 URL 2.检查所访问的服务是否可用 3.在浏览器输入 API 域名看能否访问 4.协议是否正确(http/https), 是 https 要在设置里 disable 掉 SSL 整数验证 5.用 self signed 证书, 需要在设置里加证书。还不行就去看 console

collections: 可以保存请求, collections 能帮助组织和管理这些请求, 还能建文件夹
还能分享--通过导出文件或分享 postman 链接

pm.test(测试名+回调函数(断言))-写测试的函数, 发生 error 是非阻塞式工作方式(一个报错不会影响到别的测试), 可以包含多个断言

路径(path)里 :加变量名字, postman 用 value 替换他(不能删除); query 参数用键值形式增删
全局变量:设置: 在 test 里 pm.globals.set("boardID",response.id)

pm.globals.unset("variable_key") //清楚变量

使用: Params 里{{全局变量名字}} 在 test 里 pm.globals.get("boardID")

环境变量(url): 作用域比环境变量少, 方便切换不同设置, 在请求构造器和全局变量一样
设置和使用只是把 globals 换成了 environments pm.environments.set()

一般只用环境变量或者只用全局变量或者集合变量(只能用 pm.variables.get())

都可以用 pm.variables.get()

把环境变量和全局变量的 initial value 设置成 YOUR KEY 等分享时就不会分享敏感信息

bulk edit 批量更改 params 的值

工作顺序: pre-request test - 进行 HTTP 请求 - 运行测试

作用域: local <data<environment<collection<global, 优先级相反, local 最大

集合变量: 值不变, 只有一个环境

数据变量: 用不同的数据集执行一样的 request

Debug: 先 console.log 去 postman 控制台去看详细信息, 不存在时是 null

Chai: Postman 的断言库, 让断言更易读

写断言: 先解析响应体, 比如 pm.response.json(), cheerio(pm.response.text())

用 hasOwnProperty()来找出有某个特定属性的 key, 一般这个 key 是每次 send 就变

自动化测试工具: Collection runner -> Postman Monitors->Newman -> Newman+Jenkins
-> Newman+其他 CI 服务器

Postman Monitor 就是在网站上按照间隔时间自动跑测试, 还可以设置邮件提醒

但是不能导入全局变量, 之导入环境变量

Newman(nodejs 库): CLI(开发命令行)工具, 让 postman 的 json 文件可以在命令行执行的
插件

Postman API: 通过 API 和 collection 交互的一种方式(生成 API key)

从 newman 访问 postman collection 的 3 种方式:

1. 分享链接(collection 修改后要更新)
2. 2.导出 json 文件(修改后要更新)
3. 3.用 postman API 和要测试的 collection 的 ID 得到一个 URL, 可以在 cmd 里就运行 monitor 的 test (postman 更新后 cmd 测试结果也同步更新)

在 newman 里使用环境变量: 先把环境变量导出成 json 文件, 在跑 cmd 里 Newman url 后面加上--environment 和 json 文件的路径

Jenkins 就是在 config 里输入 command--newman run, 然后指定环境和 reporters 和报告位置来生成报告, 点 save 然后点 build 来运行, 如果公布了报告+下载了 plugin(html,

Junit)(在 post-build action 里),就能看到测试的更详细的技术信息。Postman 里更新也会同步更新测试结果

gitlab-runner(项目自动话驱动)可以执行一个 YAML 文件, 当 push 代码后会执行写好的脚本, 由脚本执行进一步操作, 从而实现自动话

Timcity 分成构建代理(构建都是通过代理服务执行)+网页版控制端(通过网页进行管理), 其他和 Jenkins 差不多

Docker(CLI): 可以虚拟化的容器技术, 让电脑保持原始状态, 可以用虚拟的方式安装 jenkins, 不会被这种软件污染电脑。跑 newman 时候把 postman 的 collection 和环境变量导出到桌面, 再在 docker 里运行

用 Git 管理 postman collection: 做修改-导出文件-commit 文件-push 修改

其他人: pull 修改-导入文件-做修改-导出文件-commit-push

Git 优势: 免费, 有修改历史, 所有数据保存在本地, 可以 code review, 修改少时候好

劣势: 手动流程, 不容易 review json 文件

3. 人工智能

1. 先安装 Tensorflow 再利用 keras 的 API 建模, 再加 layer, 训练, 再用测试的数据来测试模型, 还有一个预测类的方法用来显示图片在每个类的概率

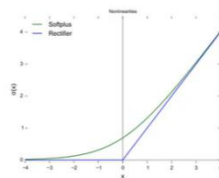
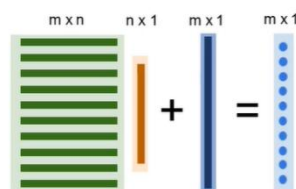
2. 先安装 PyTorch, 用 forward、backward、loss computation 一些方法训练模型, 同样加了不同种类的层比如 Flatten、Dense 层 ReLU activation 函数(来自 torch API)

Tensor 类似 numpy 数组用来解读模型的输入和输出但可以在 GPU 上运行深度学习: 利用多层次的分析和计算手段得到结果的一种方法

把一个函数分解成非线性函数的集合, 每个函数代表使用一层神经网络

- General form for each layer $a = \sigma(\underline{W^T x + b}) = g(x; W, b)$
- σ the activation function
- **Key element:** Linear operations + Nonlinear activations

- **Key element:** Linear operations + Nonlinear activations $\sigma(\underline{W^T x} + \underline{b})$



activation 函数有好几种选择

线性函数可以选 convolution(滑动窗口, 每个窗口一个结果, 最后得到一个总结, 比如一个图像, 好处: 稀疏连接, 每个输出只取决于少量输入)

pooling 增大感受野, 让 convolution 看到更多信息, 但在降维过程中丢了一些信

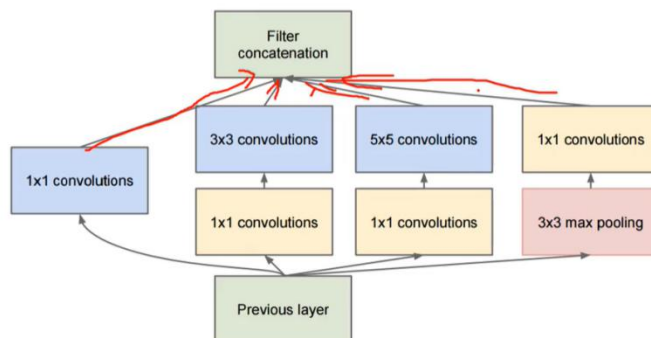
息--降低了分辨率

从对简单的图像进行分类

MNIST 数据集：7 万张照片，6 训练，1 万测试神经网络

Case study: GoogLeNet (2014)

- The Inception Module
 - Parallel paths with different receptive field sizes and operations
 - Use 1x1 convolutions for dimensionality reduction before expensive convolutions



4. 爬虫

把数据录入到 csv 中,有 6 个 section,再把数据从所有 csv 文件里拿出来按 section 整合成 6 块, 再对数据进行格式处理

```
{'data': [{'Title': 'A horror story: India Covid deaths pass 200,000',
  'Tag': 'Asia',
  'Summary': 'There are fears the real number is even higher, as hospitals continue to turn away dying patients away.',
  'URL': 'https://www.bbc.com/news/world-asia-56919924',
  'Date': '2021.04.28',
  'Index': 0},
  {'Title': 'Man charged over oxygen SOS for dying grandfather',
  'Tag': 'India',
  'Summary': 'None',
  'URL': 'https://www.bbc.com/news/world-asia-india-56894757',
  'Date': '2021.04.28'}
```

再把数据按 section 生成并放到 6 个 json 文件, 再对每个 json 文件创建一个表把 json 里面的数据插入到表中, 这些数据就都到 db 文件里了

再通过 terminal 把数据生成.sql --- 从 sqlite 导入到 MySQL source+BBC_NEWS.sql 到路径下--sqlite3--.open BBC_NEWS.db--.output BBE_NEWS.sql--.dump

MySQL 里输入指令要加分号;

- 1.反爬机制: 1.设置限制访问: 一段时间内一个 IP 不能访问
- 2.返回验证码: 要求用户输入验证码, 防的是机器
- 3.爬虫陷阱: 让爬取的信息和本网站无关

解决反爬机制: 设置不同 IP 代理这样相当于不同用户在爬网站

首先去神龙 http 网站获取大量 IP 比如 115.208.46.91:35508

再从中随机取去爬, 如果有验证机制, 每爬完一页设置程序暂停时间

困难: 把 title 和 summary 插入到表的时候发现一直有 Error: near "A":syntax error,

看了好几遍建表和插入数据的语法没有错误

解决：上网搜发现可能是 **title** 和 **summary** 的内容里有引号导致的，就把里面引号用 **replace** 去掉